

UNIT09:SPRITE – BATCHING

【学習要項】

- ☐Optimaizations
- ☐Batching
- ☐Primitive topology

【演習手順】

1. sprite_batch.h, sprite_batch.cpp をプロジェクトに新規作成で追加する
2. sprite.h の内容をすべて sprite_batch.h にコピーする
3. sprite.cpp の内容をすべて sprite_batch.cpp にコピーする
4. sprite_batch.h を編集する
 - ①クラス名を sprite_batch に変更する
 - ②コンストラクタ名と引数を変更する

```
sprite_batch(ID3D11Device *device, const wchar_t* filename, size_t max_sprites);
```

- ③デストラクタ名を変更する

- ④メンバ変数を追加する

※必要なヘッダファイルをインクルードする

```
const size_t max_vertices;  
std::vector<vertex> vertices;
```

- ⑤メンバ関数を追加する

```
void begin(ID3D11DeviceContext* immediate_context);  
void end(ID3D11DeviceContext* immediate_context);
```

5. sprite_batch.cpp を編集する

※すべてのクラス名を sprite から sparite_batch に変更する

※インクルードするファイルを sprite.h から sparite_batch.h に変更する

- ①コンストラクタの変更

```
1:  sprite_batch::sprite_batch(ID3D11Device* device, const wchar_t* filename, size_t max_sprites)  
*2:      : max_vertices(max_sprites * 6)  
3:  {  
4:      HRESULT hr{ S_OK };  
5:  
*6:      //vertex vertices[]  
*7:      //{  
*8:      // { { -1.0, +1.0, 0 }, { 1, 1, 1, 1 }, { 0, 0 } },  
*9:      // { { +1.0, +1.0, 0 }, { 1, 1, 1, 1 }, { 1, 0 } },  
*10:     // { { -1.0, -1.0, 0 }, { 1, 1, 1, 1 }, { 0, 1 } },  
*11:     // { { +1.0, -1.0, 0 }, { 1, 1, 1, 1 }, { 1, 1 } },  
*12:     //};  
*13:     std::unique_ptr<vertex[]> vertices{ std::make_unique<vertex[]>(max_vertices) };  
14:  
15:     D3D11_BUFFER_DESC buffer_desc{};  
*16:     buffer_desc.ByteWidth = sizeof(vertex) * max_vertices;  
17:     buffer_desc.Usage = D3D11_USAGE_DYNAMIC;  
18:     buffer_desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;  
19:     buffer_desc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;  
20:     buffer_desc.MiscFlags = 0;  
21:     buffer_desc.StructureByteStride = 0;  
22:     D3D11_SUBRESOURCE_DATA subresource_data{};  
*23:     subresource_data.pSysMem = vertices.get();  
24:     subresource_data.SysMemPitch = 0;  
25:     subresource_data.SysMemSlicePitch = 0;  
26:     hr = device->CreateBuffer(&buffer_desc, &subresource_data, &vertex_buffer);  
27:     _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));  
28:     :  
29:     : 省略  
30:     :  
31: }
```

②render メンバ変数の変更

※4 頂点の NDC における位置が算出できた後に、以下のコードを追加する

※それ以降のコード（頂点バッファの更新、各ステートのバインド、ドローコール）は end メンバ関数に移動する

```

1: void sprite_batch::render(ID3D11DeviceContext* immediate_context,
2:   float dx, float dy, float dw, float dh,
3:   float r, float g, float b, float a,
4:   float angle/*degree*/,
5:   float sx, float sy, float sw, float sh)
6: {
7:     :
8:     :   省略
9:     :
10:    // Convert to NDC space
11:    x0 = 2.0f * x0 / viewport.Width - 1.0f;
12:    y0 = 1.0f - 2.0f * y0 / viewport.Height;
13:    x1 = 2.0f * x1 / viewport.Width - 1.0f;
14:    y1 = 1.0f - 2.0f * y1 / viewport.Height;
15:    x2 = 2.0f * x2 / viewport.Width - 1.0f;
16:    y2 = 1.0f - 2.0f * y2 / viewport.Height;
17:    x3 = 2.0f * x3 / viewport.Width - 1.0f;
18:    y3 = 1.0f - 2.0f * y3 / viewport.Height;
19:
*20:    float u0{ sx / texture2d_desc.Width };
*21:    float v0{ sy / texture2d_desc.Height };
*22:    float u1{ (sx + sw) / texture2d_desc.Width };
*23:    float v1{ (sy + sh) / texture2d_desc.Height };
24:
*25:    vertices.push_back({ { x0, y0 , 0 }, { r, g, b, a }, { u0, v0 } });
*26:    vertices.push_back({ { x1, y1 , 0 }, { r, g, b, a }, { u1, v0 } });
*27:    vertices.push_back({ { x2, y2 , 0 }, { r, g, b, a }, { u0, v1 } });
*28:    vertices.push_back({ { x2, y2 , 0 }, { r, g, b, a }, { u0, v1 } });
*29:    vertices.push_back({ { x1, y1 , 0 }, { r, g, b, a }, { u1, v0 } });
*30:    vertices.push_back({ { x3, y3 , 0 }, { r, g, b, a }, { u1, v1 } });
*31: }

```

③begin メンバ変数の実装

```

1: void sprite_batch::begin(ID3D11DeviceContext* immediate_context)
2: {
3:     vertices.clear();
4:     immediate_context->VSSetShader(vertex_shader, nullptr, 0);
5:     immediate_context->PSSetShader(pixel_shader, nullptr, 0);
6:     immediate_context->PSSetShaderResources(0, 1, &shader_resource_view);
7: }

```

④end メンバ変数の実装

※render メンバ関数から移動したコードを下記の通りに編集する

```

1: void sprite_batch::end(ID3D11DeviceContext* immediate_context)
2: {
3:     HRESULT hr{ S_OK };
4:     D3D11_MAPPED_SUBRESOURCE mapped_subresource{};
5:     hr = immediate_context->Map(vertex_buffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mapped_subresource);
6:     _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
7:
* 8:     size_t vertex_count = vertices.size();
* 9:     _ASSERT_EXPR(max_vertices >= vertex_count, "Buffer overflow");
*10:    vertex* data{ reinterpret_cast<vertex*>(mapped_subresource.pData) };
*11:    if (data != nullptr)
*12:    {
*13:        const vertex* p = vertices.data();
*14:        memcpy_s(data, max_vertices * sizeof(vertex), p, vertex_count * sizeof(vertex));
*15:    }
16:    immediate_context->Unmap(vertex_buffer, 0);

```

```
17:
18:     UINT stride{ sizeof(vertex) };
19:     UINT offset{ 0 };
20:     immediate_context->IASetVertexBuffers(0, 1, &vertex_buffer, &stride, &offset);
*21:     immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
22:     immediate_context->IASetInputLayout(input_layout);
23:
*24:     immediate_context->Draw(static_cast<UINT>(vertex_count), 0);
25: }
```

6. パフォーマンステスト

- ① framework クラスのメンバ変数として `sprite_batch`*型配列を要素数 8 で宣言する

```
sprite_batch* sprite_batches[8];
```

- ② framework クラスの `initialize` メンバ関数で `sprite` オブジェクトを生成する

※今回は先頭の 1 個だけを生成する

```
sprite_batches[0] = new sprite_batch(device, L"¥¥resources¥¥player-sprites.png", 2048);
```

- ③ framework クラスの `render` メンバ関数に下記テストコードを入力する

※マクロのフラグを切り替えて `sprite` と `sprite_batch` の実行速度(FPS)の変化を確認する

```
1:     float x{ 0 };
2:     float y{ 0 };
3:     #if 0
4:     for (size_t i = 0; i < 1092; ++i)
5:     {
6:         sprites[1]->render(immediate_context,
7:             x, static_cast<float>(static_cast<int>(y) % 720), 64, 64,
8:             1, 1, 1, 1, 0, 140 * 0, 240 * 0, 140, 240);
9:         x += 32;
10:        if (x > 1280 - 64)
11:        {
12:            x = 0;
13:            y += 24;
14:        }
15:    }
16: #else
17:     sprite_batches[0]->begin(immediate_context);
18:     for (size_t i = 0; i < 1092; ++i)
19:     {
20:         sprite_batches[0]->render(immediate_context,
21:             x, static_cast<float>(static_cast<int>(y) % 720), 64, 64,
22:             1, 1, 1, 1, 0, 140 * 0, 240 * 0, 140, 240);
23:         x += 32;
24:         if (x > 1280 - 64)
25:         {
26:             x = 0;
27:             y += 24;
28:         }
29:     }
30:     sprite_batches[0]->end(immediate_context);
31: #endif
```

7. 最適化

※`sprite`, `sprite_batch` クラスの `render` メンバ関数内における、`rotate` ラムダ式がボトルネックになっている

- ①関数内での三角関数の処理を外部に移す

- ②ラムダ式からインライン関数に変える（関数を呼び出さない形に書き下してもよい）

【評価項目】

□スプライトバッチング